Cooperative Cloud SLAM on Matrix Lie Groups

Kruno Lenac, Josip Ćesić, Ivan Marković, and Ivan Petrović

University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
{kruno.lenac, josip.cesic, ivan.markovic, ivan.petrovic}@fer.hr

Abstract. In this paper we present a Cooperative Cloud SLAM on Matrix Lie Groups (C^{2}LEARS), which enables efficient and accurate execution of simultaneous localization and environment mapping, while relying on integration of data from multiple agents. Such fused information is then used to increase mapping accuracy of every agent itself. In particular, the agents perform only computationally simpler tasks including local map building and single trajectory optimization. At the same time, the efficient execution is ensured by performing complex tasks of global map building and multiple trajectory optimization on a standalone cloud server. The front-end part of C^{2}LEARS is based on a planar SLAM solution, while the back-end is implemented using the exactly sparse delayed state filter on matrix Lie groups (LG-ESDSF). The main advantages of the front-end employing planar surfaces to represent the environment are significantly lower memory requirements and possibility of the efficient map exchange between agents. The back-end relying on the LG-ESDSF allows for efficient trajectory optimization utilizing sparsity of the information form and exploiting higher accuracy supported by representing the state on Lie groups. We demonstrate C^{2}LEARS on a real-world experiment recorded on the ground floor of our faculty building.

Keywords: cooperative SLAM, cloud SLAM, exactly sparse delayed state filter, Lie groups, planar map

1 Introduction

The Simultaneous Localization and Mapping (SLAM) [1] is an essential component of every autonomous mobile robot. It gives robot the ability to know its location and build the map of its environment simultaneously. All modern SLAM algorithms have two key components: (i) a front-end and (ii) a back-end. The front-end is responsible for processing data from the sensors, detecting loop closings, and finding pose constraints. The back-end is responsible for optimization of both landmarks and robot poses. In general, SLAM is a passive solution that does not alter robot's movement. Instead it continuously works in the background and provides necessary data like location to higher level algorithms, e.g., exploration, navigation etc. In many cases these higher level algorithms would benefit from multiple agent cooperation in order to complete their tasks faster. For example, when considering the exploration task, which requires building a complete map of the area in the shortest possible time, several agents (e.g. ground and aerial robots) would complete the task much faster than a single agent. The easiest way to do this would be to explore a part of the environment with each agent using one of the available single agent SLAM algorithms, and then merge local maps in a single global map. However, this solution has several drawbacks, and the most important one is the inability of one agent to use information from other agents to increase its own mapping accuracy. Algorithms that overcome this problem are called cooperative SLAM algorithms.

Front-end part of the cooperative SLAM requires processing the data from sensors similarly to that of the SLAM system used for single agent. However, SLAM front-end in cooperative solution additionally has to be able to communicate with other agents and exchange sensor data. If the communication bandwidth is high, raw sensor data can be exchanged. This solution is suitable for centralized SLAM systems [2, 3], since in that case agents do not need to process data, but rather send it over the network to some central processing unit. If the bandwidth is limited and agents have enough power to process the measurement data, raw data is filtered and reduced in size, and only some preprocessed information is then transmitted over the network. Examples of such approaches can be found in [4, 5].

Regardless of the front-end implementation and information sharing, the main difference between cooperative SLAM algorithms is the back-end implementation. The SLAM back-ends can be divided in two groups depending on the optimization techniques. One group contains SLAM systems that use filtering based approaches relying on the extended Kalman filter (EKF) and particle filter (PF) implementations, while another group relies on graph optimization techniques like g^{2}o [6] and iSAM2 [7]. Among all of the approaches, the easiest transformation from single to multiple agent SLAM is achieved using the EKF SLAM back-end. Implementation of a cooperative EKF-SLAM [8, 9] is then mostly straightforward. However, multiple agent SLAM based on EKF faces the same issues as the single agent EKF SLAM, including linearization errors and increased computational complexity with the increased number of landmarks. More advanced filtering based solutions for cooperative SLAM use PF and the extended information filter (EIF). The main advantage of the PF coopearative SLAM [2, 10] is its requirement to linearize only the measurement model, while its real-time computation is maintained using Rao-Blackwellization [11]. EIF for multiple agent SLAM [12] allows simpler decentralization of the information acquired by each agent, but the main problem of EKF still remains; the issue of slow computation speed when number of landmarks increases. This problem was solved in [13] where sparsification of the information matrix is used in order to maintain computational efficiency. Cooperative SLAM algorithms that use graph optimization back-ends [14, 15] generally employ global graph consisting of subgraphs built by each agent. If relative pose between agents is not known in advance, subgraphs are not connected until the agents meet, whereas if the relative pose is known, the subgraphs are connected from the beginning. A more thorough analysis of all aspects and differences between various multiple agent SLAM algorithms can be found in [16].

Solution proposed in this paper for cooperative SLAM is most similar to the one presented in [17], where authors present cooperative cloud based SLAM dubbed C^{2}TAM. C^{2}TAM is a mix between centralized and decentralized SLAM system in which each agent performs its own localization using computationally light visual odometry, while computational costly steps, including optimization and map building, are executed on an external server. The main idea behind our cooperative SLAM system is also to divide the computational load; however, there are three key differences between C^{2}LEARS and C^{2}TAM: (i) instead of bundle adjustment we use filter based back-end, (ii) each agent performs its own trajectory estimation and optimization, while also maintaining local map which allows it to operate completely independently in the case of other agent and/or server failure, and (iii) we build planar map of the environment which drastically reduces the memory and computation requirements when exchanging and using maps.

2 SLAM for a single agent

In this section we present key properties of the SLAM back-end and front-end developed for single agent SLAM as part our previous research. This represents a basis for our newly developed cooperative SLAM system C^{2}LEARS.

For the SLAM back-end we have used exactly sparse delayed state filter (ESDSF) [18] implemented using matrix Lie groups (LG-ESDSF) [19]. As demonstrated in [19] by implementing the filter using Lie groups we were able to respect state-space geometry and thus achieve significant improvement in accuracy, comparable to state-of-the-art graph optimization based SLAM solutions. Although ESDSF is a special case of EIF, and has two distinctive properties: (i) information matrix is exactly sparse which means matrix inversion can be drastically speeded up using sparse matrix solvers with no need for sparsification; (ii) state space consists of discrete trajectory states, while map consists of fused measurements acquired in those states. This means that map landmarks are independent of each other and that map estimation can be done in parallel with the trajectory estimation. As is explained in the next section, this represents the crucial property for developing C^{2}LEARS. Trajectory T_n in LG-ESDSF is represented by n discrete states X_i, $i = 1 \dots n$, where each state X_i represents agent's pose as $\mathsf{SE}(3)$ group element. The variable T_n is constructed as follows:

$$X_i = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix}, \; T_n = \begin{bmatrix} X_1 & 0 & 0 & 0 \\ 0 & X_2 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & X_n \end{bmatrix} \; \begin{array}{l} X_i \sim \mathcal{N}(\mu_{X_i}, \Sigma_{X_{i,i}}) = \mathcal{N}(\eta_{X_i}, \Lambda_{X_{i,i}}) \\ T_n \sim \mathcal{N}(\mu_n, \Sigma_n) = \mathcal{N}(\eta_n, \Lambda_n) \end{array}, \tag{1}$$

where R_i represents matrix defining the agent's orientation in the global frame, t_i represents agent's position in the global frame, μ_{X_i} and Σ_{X_i} represent mean and covariance of X_i, while μ_n and Σ_n are mean and covariance of the trajectory T_n, respectively. Similarly to classic EIF, LG-ESDSF operates by evaluating two steps; prediction and update. However, during the prediction step instead of just predicting agent's current location, the newly predicted location is also added as a new discrete state in T_n. This process is refereed to as augmentation of trajectory T_n. Once the trajectory has been augmented, the state X_n remains in the trajectory if the difference between the newly augmented state X_{n+1} and state X_n is larger than the predefined threshold; otherwise it is marginalized and T_{n+1} becomes T_n with new state X_{n+1} replacing state X_n. Motion model used to predict state X_{n+1} is given as nonlinear first order Markov process

$$X_{n+1} = f(X_n, \Omega_n, w_n)\,, \tag{2}$$

where the control signal Ω_n represents change in the agent's pose between X_n and X_{n+1} measured by odometry, while w_n represents the zero-mean white Gaussian noise with covariance Q_n.

If loop closing is detected between newly augmented state X_{n+1} and state X_i, measurements taken during the augmentation are sent to the Relative Pose Estimation algorithm (RPE) described in [20]. Once relative pose is calculated, it is further used in LG-ESDSF to perform the trajectory update. The measurement model is defined as

$$h(T_{n+1}) = X_i^{-1} X_{n+1}, \tag{3}$$

and the measurement Jacobian evaluates to

$$\mathcal{H}_{n+1} = [\cdots\ 0_{6\times 6}\ \cdots\ -\mathrm{Ad}(X_{n+1}^{-1})\,\mathrm{Ad}(X_j)\ \cdots\ I_{6\times 6}]\,,$$

where Ad represents adjoint operator on matrix Lie groups. This result means that update step is always constant time. Since prediction step is also constant time (it changes only four blocks in the information matrix) the only computationally expensive operation is the inversion of the information matrix. It is performed after each update step, since it is required for the re-normalization of Λ (as explained in [21]). However, as mentioned above, this step can be drastically speeded up with the usage of sparse matrix solvers.

The SLAM front-end used in C^{2}LEARS is based on our single SLAM front-end [20]. It uses point clouds generated by 3D LIDAR and segments those point clouds into planar surface segments. Whenever a new state X_n is added into the trajectory, point cloud P_n is recorded and segmented. An example of a segmented point cloud is shown in Fig. 1. All planar surface segments from one point cloud form local planar map M_n. The local planar maps are then used for two purposes: (i) when loop closing is detected between X_i and X_j, RPE uses M_i and M_j to calculate relative pose between X_i and X_j after which SLAM uses relative pose constraint to perform trajectory update, and (ii) for building the global planar map. Whenever new local map M_n is built, it is sent to the global map building module (GMB). GMB searches for pairs between the planar surface segments from the current global map and planar surface segments in the local map M_n. All segments that have a match in the global map are fused

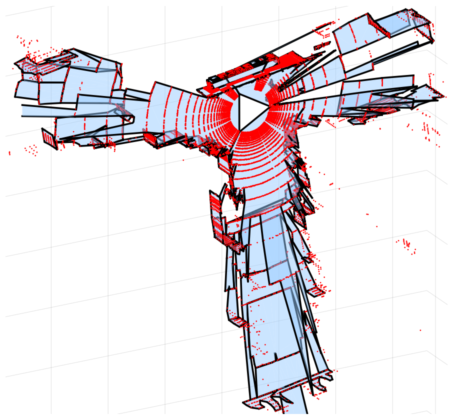

Fig. 1. Point cloud segmented into planar surface segments.

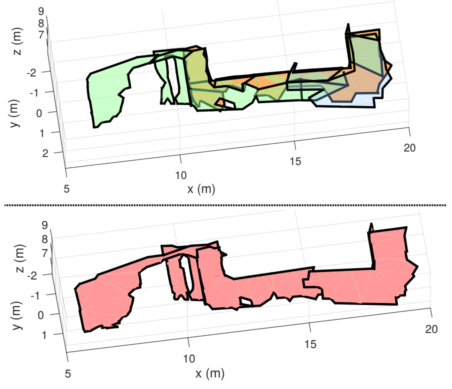

Fig. 2. Planar segments from local maps (up) merged in the global map (down).

with the existing global planar surfaces while others form a new global planar surfaces. An example of fusion of several planes between three local maps is shown in Fig. 2.

Loop closing detection between states in the trajectory is done in the similar way as in [20] and is based on the work presented in [22]. Whenever a new state X_n is added into the trajectory, the graph ${}^{T}g$ is generated from the pose graph incidence matrix ${}^{T}I$. The incidence matrix is $n \times n$ matrix whose element (i,j) has value 1 if the states X_i and X_j are neighbouring states or if the loop was closed between them. Otherwise the element value is 0. Each node in ${}^{T}g$ represents one state in trajectory, the connections are made based on ${}^{T}I$, while weights of the connections $w_{i,j}$ are given as

$$w_{i,j} = e_{trans} + \alpha e_{rot}\,, \tag{4}$$

where α is the scaling factor, and

$$\Delta T = X_i^{-1} X_j = \begin{bmatrix} & R & & \Delta x \\ & & & \Delta y \\ & & & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{aligned} e_{trans} &= \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \\ e_{rot} &= \arccos\left(\frac{\mathrm{trace}(\Delta T)}{2} - 1\right) \end{aligned} . \tag{5}$$

Once the graph is generated, topological distance between states X_i, $i < n$ and X_n is calculated for all states whose $w_{n,i}$ is smaller than the predefined threshold. Topological distance is equal to the weight of the shortest path in the graph ${}^{T}g$ from state X_i to state X_n calculated using A^* algorithm. All states whose topological distance is higher than the predefined threshold are reported for loop closing. This way only loop closings with high information gain are selected. For detailed description of surface matching algorithm, global map building and loop closing detection please confer [20].

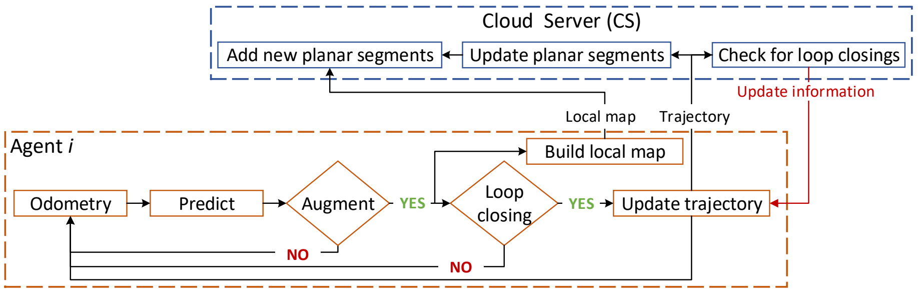

Fig. 3. Overview of the proposed cooperative SLAM solution.

3 Cooperative SLAM

Our cooperative SLAM solution was designed with the main goal we wanted to achieve being a faster exploration of indoor spaces using multiple agents. We also wanted our system to be able to: (i) operate if one of the agents included in the system fails, (ii) use the experience of some agent for improving the accuracy of another and (iii) to be able to quickly exchange maps between agents.

The main assumption of our cooperative SLAM is that relative poses between all agents are known at the beginning. Although this may seem to be a limiting factor, we believe it is acceptable respecting its intended use for faster exploration. The reasoning is that we do not need absolute starting location, but only relative poses between one reference agent and all other agents. Also, we only require agents to be on the line of sight at the very beginning, hence scans taken from the 3D LIDAR can be initially matched using the RPE algorithm. Although we acknowledge that this condition can be limiting for some tasks, in the case of our multiple agent exploration application, this requirement, for the agents to start from relatively similar locations, is easily reachable.

The overview of our proposed system is shown in Fig. 3. Each agent in the system builds its own trajectory using SLAM back-end described in Section 2. It also performs loop closings and segmentation of measurements into the planar surface segments. However, once the new local planar map M_i is built, it is sent to the Cloud Server (CS) alongside with the current trajectory of the respective agent. After CS receives M_i, it incorporates its planar segments into the global map using current trajectory information. The incorporation of local planar maps from different agents into the global map is possible based on their trajectories, since their relative poses are known and every trajectory is built within the same global frame. CS also compares the received trajectory with the last previously received trajectory of particular agent and if the change in state poses exceeds predefined threshold, global planar segments are updated accordingly.

This way, each agent has to have only enough computation power to estimate its trajectory and build local planar maps. As explained in Section 2, all steps of the LG-ESDSF have very low computational cost, which is also confirmed in the

Sec. 4 describing the experimental results. By allowing each agent to maintain its own trajectory, we have achieved robustness since in the case of another agent or CS failure, each agent will know exactly where it is. One could argue that building local planar maps on-board the agent is unnecessary and that it should also be performed by CS. However, by having local maps available on-board an agent, we benefit gaining two key advantages. First, only segmented local planar maps need to be transferred to the cloud server instead of the entire point cloud, which achieves our goal of simpler and faster map exchange. Second, each agent has its own local planar map which can be used in combination with trajectory for navigation to the predefined safe location in case CS or connection with it fails. Building local planar maps directly on-board each agent is enabled by our planar segmentation algorithm [20] which is computationally inexpensive and moreover needs to be executed only when the trajectory is augmented with new state. Besides that, segmentation process can be performed in parallel with the trajectory estimation because trajectory estimation is independent from segmenting local planar maps.

The most important advantage from knowing relative poses between agents in the beginning is the ability to improve their entire trajectories and global map every time they meet later during the environment exploration. When the initial relative pose of the agents is unknown, their trajectories will be also corrected every time they meet, but the problem is that the trajectories and global map corrections are almost entirely limited to the area of the environment that was mapped by both agents before they have met. However, since all robot states and map landmarks are connected they will all be corrected in the global frame, i.e. also outside the overlapping area, but their relative poses can only be marginally corrected until further rendezvouses occur in different locations. Since in our case of multi-agent exploration we cannot predict the moments when the agent's trajectories intersect, knowing initial relative poses is of great importance for maximally exploiting information from every meeting.

In order to use one agent's information to correct the trajectory of another, loop closing between their trajectories has to be detected. This is executed by the CS for two reasons: (i) only CS has the trajectory of every agent and (ii) in order to find the loop closing between different trajectories, trajectories from all the agents need to be checked which can be a costly computation. The CS searches for possible loop closings in the similar way as a single agent does, using the algorithm explained in Sec. 2. The example shown in Fig. 4 depicts trajectories from two agents: a (blue) and b (orange). Agents started to map the environment from poses X_0^a and X_0^b and at some point they arrived at poses X_i^a and X_j^b, respectively, after which the CS detected the loop closing. Once the pair (X_i^a, X_j^b) is identified for loop closing CS uses RPE to estimate relative pose $T_{i,a}^{j,b}$ between them. The CS then calculates uncertainties of states X_i^a and X_j^b. The trajectory which contains the state with the higher uncertainty is selected for correction based on the other trajectory. Let's say that the state X_j^b has larger uncertainty. In order to correct trajectory of the b-th agent, since every agent possesses only its own trajectory, update must be performed based on the

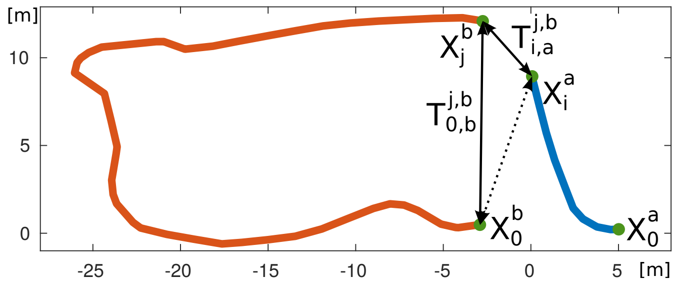

Fig. 4. Example of b-th agent's trajectory (orange) update using information from a-th agent's trajectory (blue).

relative pose between the state X_j^b and some other state X_k^b, $k \neq j$ from the same trajectory. Any state can be chosen from b-th agent's trajectory as X_k^b, but we want to choose the state with the smallest uncertainty because then the loop closing will have the largest impact on accuracy. The state that best satisfies both conditions is state X_0^b since it is the first state in the trajectory. Now in order to perform the update of b-th agent's trajectory, we have to find $T_{j,b}^{0,b}$ using new measurement $T_{i,a}^{j,b}$ and a-th agent's trajectory. Since all trajectories are within the same global frame we can do this as:

$$T_{j,b}^{0,b} = (X_0^b)^{-1}(X_i^a T_{i,a}^{j,b}). \tag{6}$$

Finally the update information is sent to the agent immediately after calculation and the update is performed right after the new state is augmented into the agent's trajectory. In general case with more than two agents, the process is repeated for every loop closing detected by the CS. Since the loop is always closed between the first trajectory state X_0 and another state X_i, it will always impact the accuracy of all states between X_j, $0 < j \leq i$. If we did not know initial relative pose between agents, this would not be possible since in equation (6) X_0^b and X_i^a would not reside in the same coordinate frame.

Since entire system is event triggered, from the SLAM perspective it is completely independent on the time synchronization. Each agent independently estimates its trajectory and planar maps, and sends them to the CS. Once the CS accepts the new local map and trajectory, it uses them to update the global map. If the new local map arrives while the CS is busy with incorporating previously received one, it will simply be incorporated as soon as CS becomes free. Currently updated global map is available at any time instant on the CS and any agent or higher level algorithm running alongside CS (e.g. exploration, navigation) can retrieve it. It is important to note that the global map is much smaller than the sum of all local maps it consists of, since it combines multiple planes from local maps into a single plane in the global map.

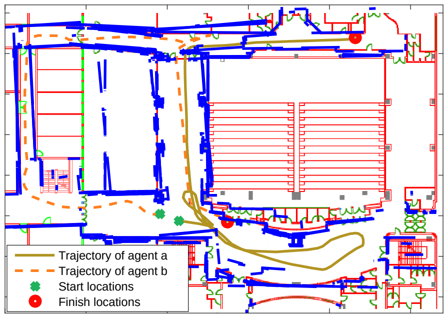

Fig. 5. Ground plan in the case when CS did not send any information.

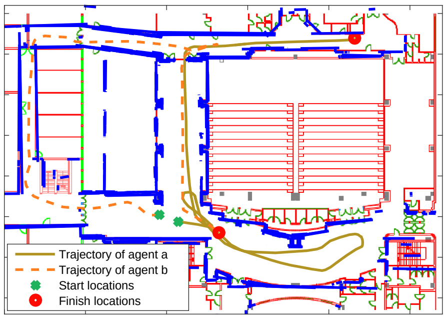

Fig. 6. Ground plan in the case when CS sent loop closing information.

4 Experimental results

In order to test our cooperative SLAM algorithm we have used the dataset we recorded for testing our the planar SLAM developed in [20]. This dataset was recorded using single mobile platform Husky A200 equipped with a Velodyne HDL-32E 3D LIDAR, while driving through the ground floor of our faculty building. All algorithms were implemented using C++ and run on portable computer Lenovo P50 with 8GB RAM and Intel Core i7@2.6 Ghz. In order to simulate cooperative behaviour we have divided dataset into two subsets. Then, we ran identical SLAM systems as separate threads simulating two agents. Each agent received data from one of the subsets. The CS was also run on the same computer as a third thread and independently received data from each agent. Interprocess communication was handled by Robotic Operating System ([23]). Since no time stamps were used in the algorithm, no generality was lost by simulating all agents and the CS on the same computer as independent threads. Odometry for prediction of both agents was calculated using point cloud matching algorithm based on Three-Dimensional Normal Distributions Transform [24].

To test the ability of one agent to improve its accuracy using information from another agent, we first performed the experiment without the CS sending loop closing information to the agents. This way every agent closed only loops detected by itself. Then we simulated the same experiment, but CS sent loop closing information to agents. Although we do not have ground-truth for trajectories, we have ground-truth for 2D ground plan of our faculty building. In order to compare results of two conducted experiments we have generated 2D ground plan by slicing the global map built by CS at certain height from the ground. We have then overlayed this 2D ground plan over the ground-truth ground plan. The results, alongside with agents' trajectories, are shown in Fig. 5 and in Fig. 6, for the case when CS did not sent anything to the agents, and for the case when CS sent loop closing information to the agents, respectively. As can be seen from the resulting figures, accuracy of the ground plan is much better in the case when CS sent information to the agents. Many duplicate planes that represent the same wall have been merged into single plane in the global map and walls have been correctly aligned.

t_{min} [ms]	t_{max} [ms]	t_{mean} [ms]
Point cloud segmentation		
58.4	166.8	115.9
Local maps matching		
3.7	13.4	8.9
LG-ESDSF update step		
0.7	3.6	2.3

Table 1. Computation times for key agent's tasks.

S_{min} [MB]	S_{max} [MB]	S_{mean} [MB]
Point clouds		
1.17	2.18	2.08
Local maps		
0.03	0.20	0.11
Local maps size [MB] / Map size [MB]		
30.2 / 2.6		

Table 2. Sizes of point clouds, resulting local maps and global map.

In Table 1 we provide minimum, maximum and mean computation times of three computationally most costly tasks performed by each agent. Those tasks are: (i) segmentation of local planar maps, (ii) their matching in case of the local loop closing and (iii) update step of LG-ESDSF. As can be seen from Table 1, matching of two local planar maps and performing update step are extremely fast and do not compromise agents ability to work in real time. As can be expected, point cloud segmentation operation is the most costly operation. However, it is still fast and as long as the agent's CPU contains at least two cores, it can be run in parallel thread with other components of the SLAM system. Table 2 shows minimum, maximum and mean sizes of point clouds and resulting local maps recorded by both agents. The segmentation of point clouds drastically reduces their size, thus allowing lower memory requirements for agents and faster transfer times over the network to the CS. Final row in Table 2 shows comparison between the cumulative size of all local maps and size of the final global map. It can also be noticed that the final map size is much smaller than the size of all local maps which allows faster transfer of the global maps and also faster computation times for the tasks which use global maps, e.g. exploration, navigation, etc. Final global map built when CS sent loop closing information is shown in 3D in Fig. 7.

5 Conclusion

In this paper we presented Cooperative SLAM solution named C^{2}LEARS. We developed the solution having three key goals in mind: (i) ability to increase mapping accuracy of one agent by using information from other agents, (ii) using compact planar map representation which allows easy map exchange between agents and server, and (iii) robustness of the entire system by allowing each agent to function independently in case of CS and/or other agent failure. To enable one agent to increase accuracy of another, we developed an algorithm which runs on the cloud server and analyses trajectories of all agents. After a suitable loop closing is located, it determines which trajectory will be updated and

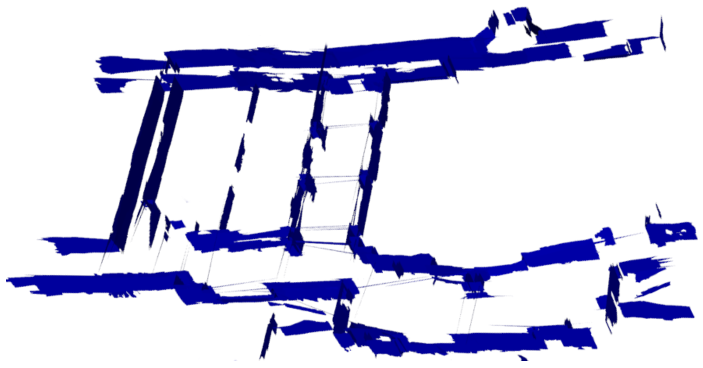

Fig. 7. Global map built by CS when loop closing information was sent to the agents.

sends the update information to the respective agent. Compact map representation was achieved by utilizing our previously developed planar SLAM algorithm which segments point clouds into planar surfaces, thus drastically reducing their size while maintaining high level of details. Finally, robustness was achieved by implementing our novel LG-ESDS filter which enables separation of trajectory estimation from global map building. This separation enabled transfer of costly global map building operation to the CS, while thanks to the sparsity of the information form and fast planar segmentation of point clouds, trajectory estimation and local map building were able to run on computationally less capable agents. We have demonstrated the effectiveness of our solution using real world dataset recorded at our faculty.

Acknowledgement

This work has been supported from the Unity Through Knowledge Fund (no. 24/15) under the project Cooperative Cloud based Simultaneous Localization and Mapping in Dynamic Environments (cloudSLAM) and has been carried out within the activities of the Centre of Research Excellence for Data Science and Cooperative Systems supported by the Ministry of Science, Education and Sports of the Republic of Croatia.

References

1. H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," *IEEE Rob. & Autom. Mag.*, vol. 2, p. 2006, 2006.
2. A. Howard, "Multi-robot simultaneous localization and mapping using particle filters," *Int. J. Rob. Res.*, vol. 25, no. 12, pp. 1243–1256, 2006.
3. B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, "Multiple relative pose graphs for robust cooperative mapping," in *ICRA*, 2010, pp. 3185–3192.
4. L. Paull, G. Huang, M. L. Seto, and J. J. Leonard, "Communication-constrained multi-auv cooperative slam." in *ICRA*, 2015, pp. 509–516.

5. A. Birk and S. Carpin, “Merging occupancy grid maps from multiple robots,” *Proc. of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006.
6. R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g2o: A general framework for graph optimization,” in *ICRA*, Shanghai, China, 2011, pp. 3607–3613.
7. M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, “iSAM2: Incremental smoothing and mapping using the Bayes tree,” *Int. J. Rob. Res.*, vol. 31, pp. 217–236, 2012.
8. A. I. Mourikis and S. I. Roumeliotis, “Predicting the performance of cooperative simultaneous localization and mapping (c-slam),” *Int. J. Rob. Res.*, vol. 25, no. 12, pp. 1273–1286, 2006.
9. X. S. Zhou and S. I. Roumeliotis, “Multi-robot slam with unknown initial correspondence: The robot rendezvous case,” in *IROS*, 2006, pp. 1785–1792.
10. L. Carlone, M. K. Ng, J. Du, B. Bona, and M. Indri, “Rao-blackwellized particle filters multi robot slam with unknown initial correspondences and limited communication,” in *ICRA*, 2010, pp. 243–249.
11. M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit, “Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges,” in *Int. Joint Conf. on AI.* Morgan Kaufmann Publishers Inc., 2003, pp. 1151–1156.
12. E. W. Nettleton, H. F. Durrant-Whyte, P. W. Gibbens, and A. H. Goektogan, “Multiple-platform localization and map building,” vol. 4196, 2000, pp. 337–347.
13. S. Thrun and Y. Liu, “Multi-robot slam with sparse-extended information filters.” *Springer Tracts in Advanced Robotics*, vol. 15, pp. 254–266, 2005.
14. A. Cunningham, K. M. Wurm, W. Burgard, and F. Dellaert, “Fully distributed scalable smoothing and mapping with robust multi-robot data association,” in *ICRA*, 2012, pp. 1093–1100.
15. V. Indelman, E. Nelson, N. Michael, and F. Dellaert, “Multi-robot pose graph localization and data association from unknown initial relative poses via expectation maximization,” in *ICRA*, 2014, pp. 593–600.
16. S. Saeedi, M. Trentini, M. Seto, and H. Li, “Multiple-robot simultaneous localization and mapping: A review,” *J. Field Robot.*, vol. 33, no. 1, pp. 3–46, 2016.
17. L. Riazuelo, J. Civera, and J. Montiel, “C2tam: A cloud framework for cooperative tracking and mapping,” *Rob. and Auton. Sys.*, vol. 62, no. 4, pp. 401 – 413, 2014.
18. M. R. Walter, R. M. Eustice, and J. J. Leonard, “Exactly Sparse Extended Information Filters for Feature-based SLAM,” *Int. J. Rob. Res.*, vol. 26, no. 4, pp. 335–359, 2007.
19. K. Lenac, J. Ćesić, I. Marković, I. Cvišić, and I. Petrović, “Revival of filtering based SLAM? Exactly sparse delayed state filter on Lie groups. ,” in *IROS*, (accepted) 2017.
20. K. Lenac, A. Kitanov, R. Cupec, and I. Petrović, “Fast planar surface 3d SLAM using LIDAR,” *Rob. and Auton. Sys.*, vol. 92, pp. 197 – 220, 2017.
21. J. Ćesić, I. Marković, M. Bukal, and I. Petrović, “Extended information filter on matrix lie groups,” *Automatica*, vol. 82, pp. 226 – 234, 2017.
22. C. Stachniss, D. Hahnel, and W. Burgard, “Exploration with active loop-closing for FastSLAM,” *IROS*, vol. 2, pp. 1505–1510, 2004.
23. M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in *ICRA*, 2009.
24. T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal, “Fast and accurate scan registration through minimization of the distance between compact 3d ndt representations,” *Int. J. Rob. Res.*, vol. 31, no. 12, pp. 1377–1393, 2012.